DonorsChoose

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of funding. Right now, a large number of volunteers is needed to manually screen each submission before it's approved to be posted on the DonorsChoose.org website.

Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three main problems they need to solve:

- How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly and as efficiently as possible
- How to increase the consistency of project vetting across different volunteers to improve the experience for teachers
- How to focus volunteer time on the applications that need the most assistance

The goal of the competition is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, using the text of project descriptions as well as additional metadata about the project, teacher, and school. DonorsChoose.org can then use this information to identify projects most likely to need further review before approval.

About the DonorsChoose Data Set

The `train.csv` data set provided by DonorsChoose contains the following features:

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal code). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy`

- Literature & Writing, Social Sciences

Feature	Description
project_resource_summary	An explanation of the resources needed for the project. **Example:** • My students need hands on literacy materials to manage sensory needs!
project_essay_1	First application essay*
project_essay_2	Second application essay*
project_essay_3	Third application essay*
project_essay_4	Fourth application essay*
project_submitted_datetime	Datetime when project application was submitted. **Example:** 2016-04-28 12:43:56.245
teacher_id	A unique identifier for the teacher of the proposed project. **Example:** bdf8baa8fedef6bfeec7ae4ff1c15c56
teacher_prefix	Teacher's title. One of the following enumerated values: • nan • Dr. • Mr. • Mrs. • Ms. • Teacher.
teacher_number_of_previously_posted_projects	Number of project applications previously submitted by the same teacher. **Example:** 2

* See the section **Notes on the Essay Data** for more details about these features.

Additionally, the `resources.csv` data set provides more data about the resources required for each project. Each line in this file represents a resource required by a project:

Feature	Description
id	A `project_id` value from the `train.csv` file. **Example:** p036502
description	Desciption of the resource. **Example:** Tenor Saxophone Reeds, Box of 25
quantity	Quantity of the resource required. **Example:** 3
price	Price of the resource required. **Example:** 9.95

Note: Many projects require multiple resources. The `id` value corresponds to a `project_id` in train.csv, so you use it as a key to retrieve all resources needed for a project:

The data set contains the following label (the value you will attempt to predict):

Label	Description
project_is_approved	A binary flag indicating whether DonorsChoose approved the project. A value of `0` indicates the project was not approved, and a value of `1` indicates the project was approved.

Notes on the Essay Data

Prior to May 17, 2016, the prompts for the essays were as follows:

- __project_essay_1:__ "Introduce us to your classroom"
- __project_essay_2:__ "Tell us more about your students"
- __project_essay_3:__ "Describe how your students will use the materials you're requesting"
- __project_essay_3:__ "Close by sharing why your project will make a difference"

Starting on May 17, 2016, the number of essays was reduced from 4 to 2, and the prompts for the first 2 essays were changed to the following:

- __project_essay_1:__ "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."

your neighborhood, and your school are all helpful."

- __project_essay_2:__ "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

  For all projects with project_submitted_datetime of 2016-05-17 and later, the values of project_essay_3 and project_essay_4 will be NaN.

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

#from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

1.1 Reading Data

In [2]:

```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [ ]:

```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

In [ ]:

```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

1.2 preprocessing of project_subject_categories

1.2 preprocessing of project_subject_categories

In [3]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

project_data['clean_categories'] = cat_list
project_data.drop(['project_subject_categories'], axis=1, inplace=True)

from collections import Counter
my_counter = Counter()
for word in project_data['clean_categories'].values:
    my_counter.update(word.split())

cat_dict = dict(my_counter)
sorted_cat_dict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))
```

1.3 preprocessing of `project_subject_subcategories`

In [4]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

project_data['clean_subcategories'] = sub_cat_list
project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

sub_cat_dict = dict(my_counter)
```

```
sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))
```

Preprocessing of 'Teacher_prefix'

In [5]:

```
teacher_pre = []
for prefix in project_data['teacher_prefix'].values:
    if prefix==prefix:
        prefix = re.sub('[^A-Za-z0-9]','',prefix).lower()
        teacher_pre.append(prefix)
    else:
        teacher_pre.append(prefix)

project_data['teacher_prefix'] = teacher_pre
```

Preprocessing of project_grade_category

In [6]:

```
project_grade_cat = []
for grade in project_data['project_grade_category'].values:
    grade = grade.replace('-','_').lower()
    grade = grade.replace(' ','_').lower()
    project_grade_cat.append(grade)
project_data['project_grade_category'] = project_grade_cat
```

1.3 Text preprocessing

In [7]:

```
# merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)
```

In [ ]:

```
project_data.head(2)
```

In [ ]:

```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

In [ ]:

```
# printing some random reviews
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
print("="*50)
print(project_data['essay'].values[20000])
print("="*50)
print(project_data['essay'].values[99999])
print("="*50)
```

In [8]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [ ]:

```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

In [ ]:

```
# \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks-python/
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

In [ ]:

```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

In [9]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

Stratified Distribution between Train-Test-Cv(64-20-16)

In [ ]:

```
from sklearn.model_selection import train_test_split as tts
X_train,X_test,y_train,y_test = tts(project_data,project_data['project_is_approved'],test_size =
0.2, stratify = project_data['project_is_approved'])
X_train.drop(['project_is_approved'],axis=1,inplace=True)
X_test.drop(['project_is_approved'],axis=1,inplace=True)
#X_cv.drop(['project_is_approved'],axis=1,inplace=True)
print(X_train.shape)
print(X_test.shape)
```

In [10]:

```
X_train = pd.read_csv('X_train')
X_test = pd.read_csv('X_test')
y_train = pd.read_csv('Y_train',names = ['Unnamed:0','project_is_approved'])
y_test = pd.read_csv('Y_test',names = ['Unnamed:0','project_is_approved'])
```

In [11]:

```
project_grade_cat_train = []
for grade in X_train['project_grade_category'].values:
    grade = grade.replace('-','_').lower()
    grade = grade.replace(' ','_').lower()
    project_grade_cat_train.append(grade)
X_train['project_grade_category'] = project_grade_cat_train
```

In [12]:

```
project_grade_cat_test = []
for grade in X_test['project_grade_category'].values:
    grade = grade.replace('-','_').lower()
    grade = grade.replace(' ','_').lower()
    project_grade_cat_test.append(grade)
X_test['project_grade_category'] = project_grade_cat_test
```

1.4 Preprocessing of Essay on Training data

In [13]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays_train = []
# tqdm is for printing the status bar
for sentance in tqdm(X_train['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed_essays_train.append(sent.lower().strip())
```

```
100%|██████████| 87398/87398 [01:26<00:00, 1004.80it/s]
```

1.4 Preprocessing of Essay on Test data

In [14]:

```
preprocessed_essays_test = []
# tqdm is for printing the status bar
for sentance in tqdm(X_test['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed_essays_test.append(sent.lower().strip())
```

```
100%|██████████| 21850/21850 [00:22<00:00, 982.08it/s]
```

1.4 Preprocessing of Title on Training data

In [15]:

```
# similarly you can preprocess the titles also
preprocessed_titles_train =[]
for title in tqdm(X_train['project_title'].values):
        des = decontracted(title)
        des = des.replace("\\r",' ')
        des = des.replace('\\"',' ')
        des = des.replace('\\n',' ')
        des = re.sub('[^A-Za-z0-9]+',' ',des)
        des = ' '.join(e for e in des.split() if e.lower() not in stopwords)
        preprocessed_titles_train.append(des.lower().strip())
```

```
100%|██████████| 87398/87398 [00:04<00:00, 19212.50it/s]
```

1.4 Preprocessing of Title on Test data

In [16]:

```
preprocessed_titles_test =[]
for title in tqdm(X_test['project_title'].values):
        des = decontracted(title)
        des = des.replace("\\r",' ')
        des = des.replace('\\"',' ')
        des = des.replace('\\n',' ')
        des = re.sub('[^A-Za-z0-9]+',' ',des)
        des = ' '.join(e for e in des.split() if e.lower() not in stopwords)
        preprocessed_titles_test.append(des.lower().strip())
```

```
100%|██████████| 21850/21850 [00:00<00:00, 35751.52it/s]
```

Sentimental Analysis of Essay

In [ ]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
senti = SentimentIntensityAnalyzer()
positive_tr,positive_ts,positive_cv=[],[],[]
negative_tr ,negative_ts,negative_cv= [],[],[]
neutral_tr,neutral_ts,neutral_cv = [],[],[]
comp_tr ,comp_ts,comp_cv= [],[],[]
```

Training Data Sentiment

In [ ]:

```
for i in tqdm(X_train['essay']):
    positive_tr.append(senti.polarity_scores(i)['pos'])
    negative_tr.append(senti.polarity_scores(i)['neg'])
    neutral_tr.append(senti.polarity_scores(i)['neu'])
    comp_tr.append(senti.polarity_scores(i)['compound'])
X_train['pos'] = positive_tr
X_train['neg'] = negative_tr
X_train['neu'] = neutral_tr
X_train['comp'] = comp_tr
```

Test Data Sentiment

In [ ]:

```
for i in tqdm(X_test['essay']):
    positive_ts.append(senti.polarity_scores(i)['pos'])
    negative_ts.append(senti.polarity_scores(i)['neg'])
    neutral_ts.append(senti.polarity_scores(i)['neu'])
    comp_ts.append(senti.polarity_scores(i)['compound'])
X_test['pos'] = positive_ts
X_test['neg'] = negative_ts
X_test['neu'] = neutral_ts
X_test['comp'] = comp_ts
```

1.5 Preparing data for models

In [ ]:

```
project_data.columns
```

we are going to consider

```
       - school_state : categorical data
       - clean_categories : categorical data
       - clean_subcategories : categorical data
       - project_grade_category : categorical data
       - teacher_prefix : categorical data

       - project_title : text data
       - text : text data
       - project_resource_summary: text data (optinal)

       - quantity : numerical (optinal)
       - teacher_number_of_previously_posted_projects : numerical
       - price : numerical
```

1.5.1 Vectorizing Categorical data

- https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

<font color = 'red'> One hot encoding of categories column in train,test,and cv data

In [17]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer( lowercase=False, binary=True)
vectorizer.fit(X_train['clean_categories'].values)
categories_one_hot_train = vectorizer.transform(preprocessed_essays_train)
categories_one_hot_test = vectorizer.transform(preprocessed_essays_test)
```

```
#categories_one_hot_cv = vectorizer.transform(preprocessed_essays_cv)
print(vectorizer.get_feature_names())
print("Shape of Train matrix after one hot encodig ",categories_one_hot_train.shape)
print("Shape of Test matrix after one hot encodig ",categories_one_hot_test.shape)
#print("Shape of CV matrix after one hot encodig ",categories_one_hot_cv.shape)
```

```
['AppliedLearning', 'Care_Hunger', 'Health_Sports', 'History_Civics', 'Literacy_Language',
'Math_Science', 'Music_Arts', 'SpecialNeeds', 'Warmth']
Shape of Train matrix after one hot encodig  (87398, 9)
Shape of Test matrix after one hot encodig  (21850, 9)
```

<font color = 'red'> One hot encoding of sub categories column in train,test,and cv data

In [18]:

```
# we use count vectorizer_sub to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_sub = CountVectorizer( lowercase=False, binary=True)
vectorizer_sub.fit(X_train['clean_subcategories'].values)
sub_categories_one_hot_train = vectorizer_sub.transform(preprocessed_titles_train)
sub_categories_one_hot_test = vectorizer_sub.transform(preprocessed_titles_test)
#sub_categories_one_hot_cv = vectorizer_sub.transform(preprocessed_titles_cv)
print(vectorizer_sub.get_feature_names())
print("Shape of Train matrix after one hot encodig ",sub_categories_one_hot_train.shape)
print("Shape of Test matrix after one hot encodig ",sub_categories_one_hot_test.shape)
#print("Shape of CV matrix after one hot encodig ",sub_categories_one_hot_cv.shape)
```

```
['AppliedSciences', 'Care_Hunger', 'CharacterEducation', 'Civics_Government',
'College_CareerPrep', 'CommunityService', 'ESL', 'EarlyDevelopment', 'Economics',
'EnvironmentalScience', 'Extracurricular', 'FinancialLiteracy', 'ForeignLanguages', 'Gym_Fitness',
'Health_LifeScience', 'Health_Wellness', 'History_Geography', 'Literacy', 'Literature_Writing', 'M
athematics', 'Music', 'NutritionEducation', 'Other', 'ParentInvolvement', 'PerformingArts', 'Socia
lSciences', 'SpecialNeeds', 'TeamSports', 'VisualArts', 'Warmth']
Shape of Train matrix after one hot encodig  (87398, 30)
Shape of Test matrix after one hot encodig  (21850, 30)
```

<font color = 'red'> One hot encoding of teacher prefix column in train,test,and cv data

In [19]:

```
#https://stackoverflow.com/questions/11620914/removing-nan-values-from-an-array
#https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-valueerror-np-nan-is
-an-invalid-document
vectorizer_prefix = CountVectorizer(lowercase = False,binary = True)
vectorizer_prefix = vectorizer_prefix.fit(X_train['teacher_prefix'].values.astype('U'))
prefix_one_hot_train = vectorizer_prefix.transform(X_train['teacher_prefix'].values.astype('U'))
#prefix_one_hot_cv = vectorizer.transform(X_cv['teacher_prefix'].values.astype('U'))
prefix_one_hot_test = vectorizer_prefix.transform(X_test['teacher_prefix'].values.astype('U'))
print(vectorizer_prefix.get_feature_names())
print("Shape of matrix after one hot encoding ", prefix_one_hot_train.shape)
#print("Shape of matrix after one hot encoding ", prefix_one_hot_cv.shape)
print("Shape of matrix after one hot encoding ", prefix_one_hot_test.shape)
```

```
['dr', 'mr', 'mrs', 'ms', 'nan', 'teacher']
Shape of matrix after one hot encoding  (87398, 6)
Shape of matrix after one hot encoding  (21850, 6)
```

<font color = 'red'> One hot encoding of project grade column in train,test,and cv data

In [20]:

```
vectorizer_grade = CountVectorizer(lowercase = False,binary = True)
vectorizer_grade = vectorizer_grade.fit(X_train['project_grade_category'].values.astype('U'))
project_grade_one_hot_train = vectorizer_grade.transform(X_train['project_grade_category'].values.
astype('U'))
#project_grade_one_hot_cv =
vectorizer.transform(X_cv['project_grade_category'].values.astype('U'))
project_grade_one_hot_test = vectorizer_grade.transform(X_test['project_grade_category'].values.ast
ype('U'))
print(vectorizer_grade.get_feature_names())
print("Shape of matrix after one hot encoding ", project_grade_one_hot_train.shape)
#print("Shape of matrix after one hot encoding ", project_grade_one_hot_cv.shape)
print("Shape of matrix after one hot encoding ", project_grade_one_hot_test.shape)
```

```
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
Shape of matrix after one hot encoding  (87398, 4)
Shape of matrix after one hot encoding  (21850, 4)
```

<font color = 'red'> One hot encoding of project grade column in train,test,and cv data

In [21]:

```
vectorizer_state = CountVectorizer(lowercase = False,binary = True)
vectorizer_state.fit(X_train['school_state'].values)
state_one_hot_train = vectorizer_state.transform(X_train['school_state'].values)
state_one_hot_test =  vectorizer_state.transform(X_test['school_state'].values)
#state_one_hot_cv = vectorizer.transform(X_cv['school_state'].values)
print(vectorizer_state.get_feature_names())
print("Shape of Train matrix after one hot encoding ", state_one_hot_train.shape)
print("Shape of Test matrix after one hot encoding ", state_one_hot_test.shape)
#print("Shape of cv matrix after one hot encoding ", state_one_hot_cv.shape)
```

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'IA', 'ID', 'IL', 'IN', 'K
S', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM',
'NV', 'NY', 'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV
', 'WY']
Shape of Train matrix after one hot encoding  (87398, 51)
Shape of Test matrix after one hot encoding  (21850, 51)
```

Essay and Title Words Count

Train Data

In [22]:

```
essay_word_counter_train = []
title_word_counter_train = []
for sent in preprocessed_essays_train:
    count = len(set(sent.split()))
    essay_word_counter_train.append(count)
for title in preprocessed_titles_train:
    count = len(set(title.split()))
    title_word_counter_train.append(count)
X_train['Essay_word_count'] = essay_word_counter_train
X_train['Title_word_count'] = title_word_counter_train
```

Test Data

In [23]:

```
essay_word_counter_test = []
title_word_counter_test = []
for sent in preprocessed_essays_test:
    count = len(set(sent.split()))
    essay_word_counter_test.append(count)
for title in preprocessed_titles_test:
    count = len(set(title.split()))
    title_word_counter_test.append(count)
X_test['Essay_word_count'] = essay_word_counter_test
X_test['Title_word_count'] = title_word_counter_test
```

1.5.3 Vectorizing Numerical features

In [24]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
X_train = pd.merge(X_train, price_data, on='id', how='left')
#X_cv = pd.merge(X_cv,price_data, on ='id',how = 'left')
X_test = pd.merge(X_test,price_data, on ='id',how = 'left')
```

Price

In [25]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-
learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler,Normalizer

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.
73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

price_scalar = Normalizer()
price_scalar.fit(X_train['price'].values.reshape(1,-1)) # finding the mean and standard deviation
of this data
#print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
price_standardized_train = price_scalar.transform(X_train['price'].values.reshape(1, -1)).reshape(-
1,1)
#price_standardized_cv = price_scalar.transform(X_cv['price'][0:12000].values.reshape(-1,1))
price_standardized_test = price_scalar.transform(X_test['price'].values.reshape(1,-1)).reshape(-1,1
)
```

Quantity

In [26]:

```
# standardized quantity columns
quantity_scaler = Normalizer()
quantity_scaler.fit(X_train['quantity'].values.reshape(1,-1))
#print(f"Mean :{quantity_scaler.mean_[0]},Standard Deviation :{np.sqrt(quantity_scaler.var_[0])}")
quantity_standardized_train = quantity_scaler.transform(X_train['quantity'].values.reshape(1,-1)).r
eshape(-1,1)
#quantity_standardized_cv = quantity_scaler.transform(X_cv['quantity'][0:12000].values.reshape(-1,
1))
quantity_standardized_test = quantity_scaler.transform(X_test['quantity'].values.reshape(1,-1)).res
hape(-1,1)
```

No.of previously done Project

In [27]:

```
#standardized projects proposed by teachers
project_scaler = Normalizer()
project_scaler.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
#print(f"Mean :{project_scaler.mean_[0]},Standard Deviation :{np.sqrt(project_scaler.var_[0])}")
project_standardized_train =
project_scaler.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,
-1)).reshape(-1,1)
#project_standardized_cv =
project_scaler.transform(X_cv['teacher_number_of_previously_posted_projects']
[0:12000].values.reshape(-1,1))
project_standardized_test =
project_scaler.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-1
)).reshape(-1,1)
```

Essay Count

In [28]:

```
#standardized Essay Count
Essay_count_scaler = Normalizer()
Essay_count_scaler.fit(X_train['Essay_word_count'].values.reshape(1,-1))
#print(f"Mean :{Essay_count_scaler.mean_[0]},Standard Deviation :
{np.sqrt(Essay_count_scaler.var_[0])}")
Essay_count_standardized_train = Essay_count_scaler.transform(X_train['Essay_word_count'].values.r
eshape(1,-1)).reshape(-1,1)
Essay_count_standardized_test =
Essay_count_scaler.transform(X_test['Essay_word_count'].values.reshape(1,-1)).reshape(-1,1)
#Essay_count_standardized_cv = Essay_count_scaler.transform(X_cv['Essay_word_count']
[:45000].values.reshape(-1,1))
```

Title Count

In [29]:

```
#standardized Title Count
title_count_scaler = Normalizer()
title_count_scaler.fit(X_train['Title_word_count'].values.reshape(1,-1))
#print(f"Mean :{title_count_scaler.mean_[0]},Standard Deviation :
{np.sqrt(title_count_scaler.var_[0])}")
title_count_standardized_train = title_count_scaler.transform(X_train['Title_word_count'].values.r
eshape(1,-1)).reshape(-1,1)
title_count_standardized_test =
title_count_scaler.transform(X_test['Title_word_count'].values.reshape(1,-1)).reshape(-1,1)
#title_count_standardized_cv = title_count_scaler.transform(X_cv['Title_word_count']
[:45000].values.reshape(-1,1))
```

Essay positive Sentiment

In [30]:

```
# normalize positive sentiment of essay
pos_senti_scaler = Normalizer()
pos_senti_scaler.fit(X_train['pos'].values.reshape(1,-1))
essay_pos_train = pos_senti_scaler.transform(X_train['pos'].values.reshape(1,-1)).reshape(-1,1)
essay_pos_test = pos_senti_scaler.transform(X_test['pos'].values.reshape(1,-1)).reshape(-1,1)
```

Essay Negative Sentiment

In [31]:

```
neg_senti_scaler = Normalizer()
neg_senti_scaler.fit(X_train['neg'].values.reshape(1,-1))
essay_neg_train = neg_senti_scaler.transform(X_train['neg'].values.reshape(1,-1)).reshape(-1,1)
essay_neg_test = neg_senti_scaler.transform(X_test['neg'].values.reshape(1,-1)).reshape(-1,1)
```

Essay Neutral Sentiment

In [32]:

```
neu_senti_scaler = Normalizer()
neu_senti_scaler.fit(X_train['neu'].values.reshape(1,-1))
essay_neu_train = neu_senti_scaler.transform(X_train['neu'].values.reshape(1,-1)).reshape(-1,1)
essay_neu_test = neu_senti_scaler.transform(X_test['neu'].values.reshape(1,-1)).reshape(-1,1)
```

Essay Compound Sentiment

In [33]:

```
comp_senti_scaler = Normalizer()
comp_senti_scaler.fit(X_train['comp'].values.reshape(1,-1))
essay_comp_train = comp_senti_scaler.transform(X_train['comp'].values.reshape(1,-1)).reshape(-1,1)
essay_comp_test = comp_senti_scaler.transform(X_test['comp'].values.reshape(1,-1)).reshape(-1,1)
```

Assignment 11: TruncatedSVD

- step 1 Select the top 2k words from essay text and project_title (concatinate essay text with project title and then find the top 2k words) based on their `idf_` values
- step 2 Compute the co-occurance matrix with these 2k words, with window size=5 (ref)
- step 3 Use TruncatedSVD on calculated co-occurance matrix and reduce its dimensions, choose the number of components (`n_components`) using elbow method

  > - The shape of the matrix after TruncatedSVD will be 2000*n, i.e. each row represents a vector form of the corresponding word.
  > - Vectorize the essay text and project titles using these word vectors. (while vectorizing, do ignore all the words which are not in top 2k words)

- step 4 Concatenate these truncatedSVD matrix, with the matrix with features
  - **school_state** : categorical data
  - **clean_categories** : categorical data
  - **clean_subcategories** : categorical data
  - **project_grade_category** :categorical data
  - **teacher_prefix** : categorical data
  - **quantity** : numerical data
  - **teacher_number_of_previously_posted_projects** : numerical data
  - **price** : numerical data
  - **sentiment score's of each of the essay** : numerical data
  - **number of words in the title** : numerical data
  - **number of words in the combine essays** : numerical data
  - **word vectors calculated in step 3** : numerical data
- step 5: Apply GBDT on matrix that was formed in step 4 of this assignment, **DO REFER THIS BLOG: XGBOOST DMATRIX**
- **step 6:Hyper parameter tuning (Consider any two hyper parameters)**
  - **Find the best hyper parameter which will give the maximum AUC value**
  - **Find the best hyper paramter using k-fold cross validation or simple cross validation data**
  - **Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning**

In [34]:

```
import sys
import math

import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score

# you might need to install this one
import xgboost as xgb
```

```
class XGBoostClassifier():
    def __init__(self, num_boost_round=10, **params):
        self.clf = None
        self.num_boost_round = num_boost_round
        self.params = params
        self.params.update({'objective': 'multi:softprob'})

    def fit(self, X, y, num_boost_round=None):
        num_boost_round = num_boost_round or self.num_boost_round
        self.label2num = {label: i for i, label in enumerate(sorted(set(y)))}
        dtrain = xgb.DMatrix(X, label=[self.label2num[label] for label in y])
        self.clf = xgb.train(params=self.params, dtrain=dtrain, num_boost_round=num_boost_round, ve
rbose_eval=1)

    def predict(self, X):
        num2label = {i: label for label, i in self.label2num.items()}
        Y = self.predict_proba(X)
        y = np.argmax(Y, axis=1)
        return np.array([num2label[i] for i in y])

    def predict_proba(self, X):
        dtest = xgb.DMatrix(X)
        return self.clf.predict(dtest)

    def score(self, X, y):
        Y = self.predict_proba(X)[:,1]
        return roc_auc_score(y, Y)

    def get_params(self, deep=True):
        return self.params

    def set_params(self, **params):
        if 'num_boost_round' in params:
            self.num_boost_round = params.pop('num_boost_round')
        if 'objective' in params:
            del params['objective']
        self.params.update(params)
        return self
'''clf = XGBoostClassifier(eval_metric = 'auc', num_class = 2, nthread = 4,)
###################################################################
#               Change from here                                  #
###################################################################
parameters = {
    'num_boost_round': [100, 250, 500],
    'eta': [0.05, 0.1, 0.3],
    'max_depth': [6, 9, 12],
    'subsample': [0.9, 1.0],
    'colsample_bytree': [0.9, 1.0],
}

clf = GridSearchCV(clf, parameters)
X = np.array([[1,2], [3,4], [2,1], [4,3], [1,0], [4,5]])
Y = np.array([0, 1, 0, 1, 0, 1])
clf.fit(X, Y)

# print(clf.grid_scores_)
best_parameters, score, _ = max(clf.grid_scores_, key=lambda x: x[1])
print('score:', score)
for param_name in sorted(best_parameters.keys()):
    print("%s: %r" % (param_name, best_parameters[param_name]))
'''
```

Out[34]:

```
'clf = XGBoostClassifier(eval_metric = \'auc\', num_class = 2, nthread =
4,)\n###################################################################\n#               Change fr
om here
#\n###################################################################\nparameters = {\n
\'num_boost_round\': [100, 250, 500],\n    \'eta\': [0.05, 0.1, 0.3],\n    \'max_depth\': [6, 9,
12],\n    \'subsample\': [0.9, 1.0],\n    \'colsample_bytree\': [0.9, 1.0],\n}\n\nclf = GridSearchC
V(clf, parameters)\nX = np.array([[1,2], [3,4], [2,1], [4,3], [1,0], [4,5]])\nY = np.array([0, 1,
0, 1, 0, 1])\nclf.fit(X, Y)\n\n# print(clf.grid_scores_)\nbest_parameters, score, _ =
max(clf.grid_scores_, key=lambda x: x[1])\nprint(\'score:\', score)\nfor param_name in
sorted(best_parameters.keys()):\n    print("%s: %r" % (param_name,
best_parameters[param_name]))\n'
```

2. TruncatedSVD

2.1 Selecting top 2000 words from `essay` and `project_title`

In [35]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
total_text = []
for i in range(80000):
    total_text.append(preprocessed_essays_train[i] + preprocessed_titles_train[i])
vectorizer = TfidfVectorizer(min_df = 10,use_idf = True,stop_words=stopwords)
model_2000 = vectorizer.fit_transform(total_text)
```

In [36]:

```
top_2000 = pd.DataFrame({"feature_names":list(vectorizer.get_feature_names()),"idf_values":list(ve
ctorizer.idf_)})
```

In [37]:

```
top_2000_features = top_2000.sort_values(by=['idf_values'],ascending=False)[:2000]
```

In [38]:

```
top_2000_features[:10]
```

Out[38]:

	feature_names	idf_values
9596	nannanspreading	9.891899
10998	ponds	9.891899
779	andrew	9.891899
10992	polymer	9.891899
782	anecdotal	9.891899
6590	illusion	9.891899
9371	nannannavigating	9.891899
14999	unfolding	9.891899
9085	nannanflash	9.891899
12964	shooters	9.891899

2.2 Computing Co-occurance matrix

In [43]:

```
def get_co_occur_matrix(data, vocab, context_window=2):
    a = pd.DataFrame(np.zeros((len(vocab), len(vocab))), index=vocab, columns=vocab)
    for review in data:
        words = review.split()
        for idx in tqdm(range(len(words))):
            if a.get(words[idx]) is None:
                continue
            for i in tqdm(range(1, context_window+1)):
                if idx-i >= 0:
                    if a.get(words[idx-i]) is not None:
                        a[words[idx-i]].loc[words[idx]] = a.get(words[idx-i]).loc[words[idx]] + 1
                        a[words[idx]].loc[words[idx-i]] = a.get(words[idx]).loc[words[idx-i]] + 1
                if idx+i < len(words):
                    if a.get(words[idx+i]) is not None:
```

```
                    a[words[idx+i]].loc[words[idx]] = a.get(words[idx+i]).loc[words[idx]] + 1
                    a[words[idx]].loc[words[idx+i]] = a.get(words[idx]).loc[words[idx+i]] + 1
    np.fill_diagonal(a.values, 0)
    return a

co_matrix = get_co_occur_matrix(total_text, list(top_2000_features['feature_names'].values))
```

Dummy

In [80]:

```
get_co_occur_matrix(["abc def ijk pqr",
"pqr klm opq",
"lmn pqr xyz abc def pqr abc"],["abc", "pqr", "def"],context_window=2)/2
```

```
  0%|                                                                                              |
[00:00<?, ?it/s]
  0%|                                                                                              |
[00:00<?, ?it/s]
100%|█████████████████████████████████████████████████████████████████████████████████████| 2/2 [00:
00<00:00, 668.20it/s]
  0%|                                                                                              |
[00:00<?, ?it/s]
100%|█████████████████████████████████████████████████████████████████████████████████████| 2/2 [00:
00<00:00, 668.20it/s]
  0%|                                                                                              |
[00:00<?, ?it/s]
100%|█████████████████████████████████████████████████████████████████████████████████████| 4/4 [00:
00<00:00, 174.47it/s]
  0%|                                                                                              |
[00:00<?, ?it/s]
  0%|                                                                                              |
[00:00<?, ?it/s]
100%|█████████████████████████████████████████████████████████████████████████████████████| 3/3 [00:
00<00:00, 501.29it/s]
  0%|                                                                                              |
[00:00<?, ?it/s]
  0%|                                                                                              |
[00:00<?, ?it/s]
100%|████████████████████████████████████████████████████████████████████████████████████| 2/2
[00:00<00:00, 1002.58it/s]
  0%|                                                                                              |
[00:00<?, ?it/s]
100%|█████████████████████████████████████████████████████████████████████████████████████| 2/2 [00:
00<00:00, 667.19it/s]
  0%|                                                                                              |
[00:00<?, ?it/s]
100%|█████████████████████████████████████████████████████████████████████████████████████| 2/2 [00:
00<00:00, 668.41it/s]
  0%|                                                                                              |
[00:00<?, ?it/s]
100%|████████████████████████████████████████████████████████████████████████████████████| 2/2
[00:00<00:00, 1002.70it/s]
  0%|                                                                                              |
[00:00<?, ?it/s]
100%|█████████████████████████████████████████████████████████████████████████████████████| 7/7 [00:
00<00:00, 219.34it/s]
```

Out[80]:

	abc	pqr	def
abc	0.0	3.0	3.0
pqr	3.0	0.0	2.0
def	3.0	2.0	0.0

I was calculating twice the actual value so i have corrected it by dividing it by 2

dividing it by 2.

In [ ]:

```
co_matrix.to_csv("Co_Occurence")
```

In [ ]:

```
co_matrix.describe()
```

In [39]:

```
co_matrix = pd.read_csv("Co_Occurence")
co_matrix = co_matrix.set_index("Unnamed: 0")/2
```

2.3 Applying TruncatedSVD and Calculating Vectors for `essay` and `project_title`

In [85]:

```
from sklearn.decomposition import TruncatedSVD
model = TruncatedSVD(n_components=1999,random_state=42)
model_svd=model.fit_transform(co_matrix)
```

In [86]:

```
variance = list(np.cumsum(model.explained_variance_ratio_))
index = list(np.arange(1,2000,1))
fig = go.Figure()
fig.add_trace(go.Scatter(x = index,y = variance,name = "Variance Explained"))
fig.add_trace(go.Scatter(x = [737],y=[1],marker = dict(color =  'red'),name = "First Point with Variance 1"))
fig.update_layout(title_text = "N_Components vs Explained Variance Ratio",xaxis = dict(title = "n_components")
                  ,yaxis = dict(title = "Variance"))
```

In [41]:

```
from sklearn.decomposition import TruncatedSVD
```

```
model = TruncatedSVD(n_components=737,random_state=42)
model_svd = model.fit_transform(co_matrix)
model_svd.shape
```

Out[41]:

```
(2000, 737)
```

In [ ]:

```
model_svd[0]
```

Essay Vectorizer

Train

In [ ]:

```
top_feats = list(top_2000_features.feature_names.values)
indx = 0
essay_vectorizer_train = []
for text in tqdm(preprocessed_essays_train[:45000]):
    vect_sum = np.zeros((1,737))
    count = 0
    for word in tqdm(text.split()):
        if word in top_feats:
            indx = top_feats.index(word)
            count+=1
            vect_sum+=model_svd[indx]
    if count!=0:
        vect_sum = vect_sum/count
    else:
        vect_sum = vect_sum
    essay_vectorizer_train.append(vect_sum)
```

Test

In [ ]:

```
top_feats = list(top_2000_features.feature_names.values)
indx = 0
essay_vectorizer_test = []
for text in tqdm(preprocessed_essays_test[:15000]):
    vect_sum = np.zeros((1,737))
    count = 0
    for word in tqdm(text.split()):
        if word in top_feats:
            indx = top_feats.index(word)
            count+=1
            vect_sum+=model_svd[indx]
    if count!=0:
        vect_sum = vect_sum/count
    else:
        vect_sum = vect_sum
    essay_vectorizer_test.append(vect_sum)
```

Title Vectorizer

Train

In [ ]:

```
top_feats = list(top_2000_features.feature_names.values)
indx = 0
indx = 0
title_vectorizer_train = []
for text in tqdm(preprocessed_titles_train[:45000]):
    vect_sum = np.zeros((1,737))
    count = 0
    for word in tqdm(text.split()):
        if word in top_feats:
            indx = top_feats.index(word)
            count+=1
            vect_sum+=model_svd[indx]
    if count!=0:
        vect_sum = vect_sum/count
    else:
        vect_sum = vect_sum
    title_vectorizer_train.append(vect_sum)
```

Test

In [ ]:

```
top_feats = list(top_2000_features.feature_names.values)
indx = 0
title_vectorizer_test = []
for text in tqdm(preprocessed_titles_test[:15000]):
    vect_sum = np.zeros((1,737))
    count = 0
    for word in tqdm(text.split()):
        if word in top_feats:
            indx = top_feats.index(word)
            count+=1
            vect_sum+=model_svd[indx]
    if count!=0:
        vect_sum = vect_sum/count
    else:
        vect_sum = vect_sum
    title_vectorizer_test.append(vect_sum)
```

In [99]:

```
essay_vectorizer_train_ = []
for i in range(45000):
    essay_vectorizer_train_.append(essay_vectorizer_train[i][0])
title_vectorizer_train_ = []
for i in range(45000):
    title_vectorizer_train_.append(title_vectorizer_train[i][0])
essay_vectorizer_test_ = []
for i in range(15000):
    essay_vectorizer_test_.append(essay_vectorizer_test[i][0])
title_vectorizer_test_ = []
for i in range(15000):
    title_vectorizer_test_.append(title_vectorizer_test[i][0])
```

In [100]:

```
np.savez_compressed("vectorizer",a = essay_vectorizer_train_,b = title_vectorizer_train_,c = essay_
vectorizer_test_,d = title_vectorizer_test_)
```

In [42]:

```
data = np.load("vectorizer.npz")
```

In [43]:

```
essay_vectorizer_train_ = list(data['a'])
title_vectorizer_train_ = list(data['b'])
essay_vectorizer_test_ = list(data['c'])
title_vectorizer_test_ = list(data['d'])
```

2.4 Merge the features from step 3 and step 4

2.4 Merge the features from step 3 and step 4

In [44]:

```
from scipy.sparse import hstack
from scipy import sparse
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_tr =
hstack((categories_one_hot_train[:45000],sub_categories_one_hot_train[:45000],prefix_one_hot_train
[:45000],
project_grade_one_hot_train[:45000],state_one_hot_train[:45000],sparse.csr_matrix(price_standardize
d_train[:45000]),
        sparse.csr_matrix(quantity_standardized_train[:45000]),sparse.csr_matrix(project_standardiz
ed_train[:45000]),
               sparse.csr_matrix(Essay_count_standardized_train[:45000]),sparse.csr_matrix(title_co
unt_standardized_train[:45000])
               ,sparse.csr_matrix(essay_pos_train[:45000]),sparse.csr_matrix(essay_neg_train[:45000
]),sparse.csr_matrix(essay_neu_train[:45000]),

sparse.csr_matrix(essay_comp_train[:45000]),sparse.csr_matrix(essay_vectorizer_train_),sparse.csr_m
atrix(title_vectorizer_train_))).tocsr()

X_ts =
hstack((categories_one_hot_test[:15000],sub_categories_one_hot_test[:15000],prefix_one_hot_test[:1
5000],
project_grade_one_hot_test[:15000],state_one_hot_test[:15000],sparse.csr_matrix(price_standardized_
test[:15000]),

sparse.csr_matrix(quantity_standardized_test[:15000]),sparse.csr_matrix(project_standardized_test[
:15000]),
               sparse.csr_matrix(Essay_count_standardized_test[:15000]),sparse.csr_matrix(title_cou
nt_standardized_test[:15000])
               ,sparse.csr_matrix(essay_pos_test[:15000]),sparse.csr_matrix(essay_neg_test[:15000])
,sparse.csr_matrix(essay_neu_test[:15000]),
               sparse.csr_matrix(essay_comp_test[:15000]),sparse.csr_matrix(essay_vectorizer_test_)
,sparse.csr_matrix(title_vectorizer_test_))).tocsr()
```

2.5 Apply XGBoost on the Final Features from the above section

In [45]:

```
from xgboost import XGBClassifier
import xgboost as xgb
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
from sklearn.metrics import roc_auc_score,roc_curve,f1_score,auc
```

XGBOOST Model with num_boost_round = 5

In [73]:

```
model = XGBoostClassifier(eval_metric = 'auc', num_class = 2, nthread = 4,num_boost_round=5)
parameters = {
    #'num_boost_round':[5,10],
    'eta':[0.01,0.03,0.05,0.1,0.3],
    'gamma':[0.01,0.03,0.1,0.2,0.3],
    'max_depth': [1, 5, 10, 50, 100, 500]

}
clf = RandomizedSearchCV(model, param_distributions=parameters,cv = 2,scoring = 'roc_auc',n_jobs=-1
)
print("fitting the model")
clf.fit(X_tr, y_train[:45000]["project_is_approved"])
```

```
fitting the model
```

Out[73]:

```
RandomizedSearchCV(cv=2, error_score='raise-deprecating',
          estimator=<__main__.XGBoostClassifier object at 0x0000024E51357E80>,
          fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'eta': [0.01, 0.03, 0.05, 0.1, 0.3], 'gamma': [0.01, 0.03, 0.1, 0.2,
0.3], 'max_depth': [1, 5, 10, 50, 100, 500]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring='roc_auc', verbose=0)
```

In [74]:

```
clf.cv_results_
```

Out[74]:

```
{'mean_fit_time': array([26.07362509, 62.06782019, 57.78663015, 58.22051394, 54.55051625,
         5.82642162,  6.52407765, 18.46119523, 77.87873685, 75.26640439]),
 'std_fit_time': array([0.10422158, 0.63133299, 0.22788906, 0.0952462 , 1.00059021,
        0.62413013, 0.05609584, 0.19900155, 1.55264008, 1.65211821]),
 'mean_score_time': array([2.67287409, 1.68505299, 1.80024779, 2.09693241, 1.91439164,
        2.06979191, 2.1361326 , 1.66955149, 1.47584283, 1.40455866]),
 'std_score_time': array([0.42936051, 0.02245224, 0.24536312, 0.06831741, 0.07628429,
        0.22119391, 0.06058371, 0.0379144 , 0.06455624, 0.33733869]),
 'param_max_depth': masked_array(data=[5, 10, 10, 10, 10, 1, 1, 5, 50, 500],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_gamma': masked_array(data=[0.03, 0.1, 0.3, 0.01, 0.3, 0.2, 0.03, 0.03, 0.1, 0.2],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_eta': masked_array(data=[0.05, 0.1, 0.01, 0.01, 0.1, 0.03, 0.1, 0.1, 0.05, 0.05],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 5, 'gamma': 0.03, 'eta': 0.05},
  {'max_depth': 10, 'gamma': 0.1, 'eta': 0.1},
  {'max_depth': 10, 'gamma': 0.3, 'eta': 0.01},
  {'max_depth': 10, 'gamma': 0.01, 'eta': 0.01},
  {'max_depth': 10, 'gamma': 0.3, 'eta': 0.1},
  {'max_depth': 1, 'gamma': 0.2, 'eta': 0.03},
  {'max_depth': 1, 'gamma': 0.03, 'eta': 0.1},
  {'max_depth': 5, 'gamma': 0.03, 'eta': 0.1},
  {'max_depth': 50, 'gamma': 0.1, 'eta': 0.05},
  {'max_depth': 500, 'gamma': 0.2, 'eta': 0.05}],
 'split0_test_score': array([0.67115782, 0.6628863 , 0.66177517, 0.6617773 , 0.66288805,
        0.61842782, 0.63473893, 0.67340637, 0.61913825, 0.61544495]),
 'split1_test_score': array([0.6699321 , 0.66604167, 0.65347957, 0.65372951, 0.66247045,
        0.61522606, 0.63117197, 0.6715827 , 0.61064807, 0.61153392]),
 'mean_test_score': array([0.67054496, 0.66446398, 0.65762737, 0.6577534 , 0.66267925,
        0.61682694, 0.63295545, 0.67249454, 0.61489316, 0.61348944]),
 'std_test_score': array([0.00061286, 0.00157768, 0.0041478 , 0.0040239 , 0.0002088 ,
        0.00160088, 0.00178348, 0.00091183, 0.00424509, 0.00195551]),
 'rank_test_score': array([ 2,  3,  6,  5,  4,  8,  7,  1,  9, 10]),
 'split0_train_score': array([0.71273144, 0.83207084, 0.76716393, 0.76825713, 0.83088164,
        0.61487793, 0.6362537 , 0.72282638, 0.98926916, 0.98607783]),
 'split1_train_score': array([0.70560984, 0.82109346, 0.75518555, 0.75547822, 0.82359224,
        0.62205891, 0.63711532, 0.71420794, 0.98874075, 0.98843433]),
 'mean_train_score': array([0.70917064, 0.82658215, 0.76117474, 0.76186768, 0.82723694,
        0.61846842, 0.63668451, 0.71851716, 0.98900496, 0.98725608]),
 'std_train_score': array([0.0035608 , 0.00548869, 0.00598919, 0.00638945, 0.0036447 ,
        0.00359049, 0.00043081, 0.00430922, 0.00026421, 0.00117825])}
```

In [78]:

```
test_auc_5 = clf.cv_results_['mean_test_score']
train_auc_5 = clf.cv_results_['mean_train_score']
```

XGBOOST Model with num_boost_round = 10

In [79]:

```
model = XGBoostClassifier(eval_metric = 'auc', num_class = 2, nthread = 4,num_boost_round=10)
parameters = {
    #'num_boost_round':[5,10],
    'eta':[0.01,0.03,0.05,0.1,0.3],
    'gamma':[0.01,0.03,0.1,0.2,0.3],
    'max_depth': [1, 5, 10, 50, 100, 500]

}
clf = RandomizedSearchCV(model, param_distributions=parameters,cv = 2,scoring = 'roc_auc',n_jobs=-1
)
print("fitting the model")
clf.fit(X_tr, y_train[:45000]["project_is_approved"])
```

```
fitting the model
```

Out[79]:

```
RandomizedSearchCV(cv=2, error_score='raise-deprecating',
          estimator=<__main__.XGBoostClassifier object at 0x0000024E67639898>,
          fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'eta': [0.01, 0.03, 0.05, 0.1, 0.3], 'gamma': [0.01, 0.03, 0.1, 0.2,
0.3], 'max_depth': [1, 5, 10, 50, 100, 500]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring='roc_auc', verbose=0)
```

In [80]:

```
clf.cv_results_
```

Out[80]:

```
{'mean_fit_time': array([124.59865391, 125.93711519,  17.51143265,  16.50955248,
          4.33277369,  15.50996482,  15.79271686,   4.60551679,
          4.69961131,  85.95672047]),
 'std_fit_time': array([1.50076783, 1.53470206, 0.49119329, 0.38302612, 0.13237453,
        0.05435455, 0.05783594, 0.06855667, 0.21144402, 0.04761755]),
 'mean_score_time': array([1.66237617, 1.647699  , 1.72091305, 1.65688956, 1.78851676,
        1.48181868, 1.64764464, 1.63391614, 1.50303268, 0.969908  ]),
 'std_score_time': array([0.08008432, 0.14806616, 0.03690875, 0.06562126, 0.09000635,
        0.01321959, 0.01346385, 0.06608844, 0.04787135, 0.00049806]),
 'param_max_depth': masked_array(data=[100, 100, 5, 5, 1, 5, 5, 1, 1, 50],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_gamma': masked_array(data=[0.3, 0.01, 0.2, 0.3, 0.2, 0.2, 0.1, 0.3, 0.03, 0.2],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_eta': masked_array(data=[0.03, 0.01, 0.01, 0.03, 0.03, 0.05, 0.05, 0.05, 0.1,
                    0.1],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 100, 'gamma': 0.3, 'eta': 0.03},
  {'max_depth': 100, 'gamma': 0.01, 'eta': 0.01},
  {'max_depth': 5, 'gamma': 0.2, 'eta': 0.01},
  {'max_depth': 5, 'gamma': 0.3, 'eta': 0.03},
  {'max_depth': 1, 'gamma': 0.2, 'eta': 0.03},
  {'max_depth': 5, 'gamma': 0.2, 'eta': 0.05},
  {'max_depth': 5, 'gamma': 0.1, 'eta': 0.05},
  {'max_depth': 1, 'gamma': 0.3, 'eta': 0.05},
  {'max_depth': 1, 'gamma': 0.03, 'eta': 0.1},
  {'max_depth': 50, 'gamma': 0.2, 'eta': 0.1}],
 'split0_test_score': array([0.61585494, 0.59168987, 0.66752736, 0.670089  , 0.61842782,
        0.67115542, 0.67115782, 0.63029876, 0.63473893, 0.61817788]),
 'split1_test_score': array([0.61009537, 0.59786191, 0.66725582, 0.66956396, 0.61522606,
        0.66997051, 0.66992724, 0.62522051, 0.63117197, 0.61437076]),
 'mean_test_score': array([0.61297516, 0.59477589, 0.66739159, 0.66982648, 0.61682694,
        0.67056297, 0.67054253, 0.62775963, 0.63295545, 0.61627432]),
 'std_test_score': array([0.00287978, 0.00308602, 0.00013577, 0.00026252, 0.00160088,
        0.00059245, 0.00061529, 0.00253913, 0.00178348, 0.00190356]),
 'rank_test_score': array([ 9, 10,  4,  3,  7,  1,  2,  6,  5,  8]),
 'split0_train_score': array([0.97652901, 0.95713405, 0.70201086, 0.70873177, 0.61487793,
        0.71271657, 0.71273144, 0.62809563, 0.6362537 , 0.99724982]),
 'split1_train_score': array([0.97910547, 0.96856304, 0.6924845 , 0.70169362, 0.62205891,
        0.70559304, 0.70559884, 0.63119807, 0.63711532, 0.99908343]),
 'mean_train_score': array([0.97781724, 0.96284855, 0.69724768, 0.70521269, 0.61846842,
        0.7091548 , 0.70916514, 0.62964685, 0.63668451, 0.99816662]),
 'std_train_score': array([0.00128823, 0.0057145 , 0.00476318, 0.00351908, 0.00359049,
        0.00356177, 0.0035663 , 0.00155122, 0.00043081, 0.0009168 ])}
```

In [83]:

```
test_auc_10 = clf.cv_results_['mean_test_score']
train_auc_10 = clf.cv_results_['mean_train_score']
```

XGBOOST Model with num_boost_round = 100

In [84]:

```
model = XGBoostClassifier(eval_metric = 'auc', num_class = 2, nthread = 4,num_boost_round=100)
parameters = {
    #'num_boost_round':[5,10],
    'eta':[0.01,0.03,0.05,0.1,0.3],
    'gamma':[0.01,0.03,0.1,0.2,0.3],
    'max_depth': [1, 5, 10, 50, 100, 500]

}
clf = RandomizedSearchCV(model, param_distributions=parameters,cv = 2,scoring = 'roc_auc',n_jobs=-1
)
print("fitting the model")
clf.fit(X_tr, y_train[:45000]["project_is_approved"])
```

```
fitting the model
```

Out[84]:

```
RandomizedSearchCV(cv=2, error_score='raise-deprecating',
          estimator=<__main__.XGBoostClassifier object at 0x0000024E67A7A518>,
          fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'eta': [0.01, 0.03, 0.05, 0.1, 0.3], 'gamma': [0.01, 0.03, 0.1, 0.2,
0.3], 'max_depth': [1, 5, 10, 50, 100, 500]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring='roc_auc', verbose=0)
```

In [85]:

```
clf.cv_results_
```

Out[85]:

```
{'mean_fit_time': array([141.02153242,  42.66933811, 140.45305264,   6.61400247,
          4.10959923,  14.24887311,   5.28719938, 106.99758339,
         13.11008656,  97.00709033]),
 'std_fit_time': array([2.05303085, 0.28326118, 3.27875316, 0.59241486, 0.29770339,
        0.04239047, 0.00849545, 0.59697342, 0.11397159, 0.11491513]),
 'mean_score_time': array([1.69772649, 1.71051121, 1.74683022, 1.70695424, 1.51397717,
        1.63019538, 1.54443443, 1.60872686, 1.49999762, 1.06567669]),
 'std_score_time': array([0.03317666, 0.03192091, 0.06532526, 0.06833553, 0.08079326,
        0.02793908, 0.00650942, 0.00994551, 0.05535197, 0.13864398]),
 'param_max_depth': masked_array(data=[500, 10, 500, 1, 1, 5, 1, 100, 5, 500],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_gamma': masked_array(data=[0.1, 0.03, 0.2, 0.03, 0.2, 0.03, 0.03, 0.01, 0.2, 0.1],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_eta': masked_array(data=[0.1, 0.01, 0.1, 0.05, 0.3, 0.1, 0.03, 0.3, 0.03, 0.01],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 500, 'gamma': 0.1, 'eta': 0.1},
 'params': [{'max_depth': 500, 'gamma': 0.1, 'eta': 0.1},
  {'max_depth': 10, 'gamma': 0.03, 'eta': 0.01},
  {'max_depth': 500, 'gamma': 0.2, 'eta': 0.1},
  {'max_depth': 1, 'gamma': 0.03, 'eta': 0.05},
  {'max_depth': 1, 'gamma': 0.2, 'eta': 0.3},
  {'max_depth': 5, 'gamma': 0.03, 'eta': 0.1},
  {'max_depth': 1, 'gamma': 0.03, 'eta': 0.03},
  {'max_depth': 100, 'gamma': 0.01, 'eta': 0.3},
  {'max_depth': 5, 'gamma': 0.2, 'eta': 0.03},
  {'max_depth': 500, 'gamma': 0.1, 'eta': 0.01}],
 'split0_test_score': array([0.61781103, 0.66178469, 0.61817788, 0.63029876, 0.66706226,
        0.67340637, 0.61842782, 0.61549041, 0.67008058, 0.59239037]),
 'split1_test_score': array([0.61247935, 0.6537266 , 0.61467598, 0.62522051, 0.6626494 ,
        0.6715827 , 0.61522606, 0.61706614, 0.66956468, 0.59953634]),
 'mean_test_score': array([0.61514519, 0.65775565, 0.61642693, 0.62775963, 0.66485583,
        0.67249454, 0.61682694, 0.61627828, 0.66982263, 0.59596335]),
 'std_test_score': array([0.00266584, 0.00402904, 0.00175095, 0.00253913, 0.00220643,
        0.00091183, 0.00160088, 0.00078786, 0.00025795, 0.00357298]),
 'rank_test_score': array([ 9,  4,  7,  5,  3,  1,  6,  8,  2, 10]),
 'split0_train_score': array([0.9961661 , 0.76820583, 0.99724982, 0.62809563, 0.67131549,
        0.72282638, 0.61487793, 0.99999995, 0.7087762 , 0.9564162 ]),
 'split1_train_score': array([0.9985647 , 0.75547455, 0.99881589, 0.63119807, 0.66916847,
        0.71420794, 0.62205891, 0.99999997, 0.70169396, 0.96784656]),
 'mean_train_score': array([0.9973654 , 0.76184019, 0.99803285, 0.62964685, 0.67024198,
        0.71851716, 0.61846842, 0.99999996, 0.70523508, 0.96213138]),
 'std_train_score': array([1.19929648e-03, 6.36563632e-03, 7.83036401e-04, 1.55121683e-03,
        1.07351080e-03, 4.30921692e-03, 3.59048836e-03, 7.92752214e-09,
        3.54112159e-03, 5.71518212e-03])}
```

In [86]:

```
test_auc_100 = clf.cv_results_['mean_test_score']
train_auc_100 = clf.cv_results_['mean_train_score']
```

XGBOOST Model with num_boost_round = 250

In [89]:

```
model = XGBoostClassifier(eval_metric = 'auc', num_class = 2, nthread = 4,num_boost_round=250)
parameters = {
    #'num_boost_round':[5,10],
    'eta':[0.01,0.03,0.05,0.1,0.3],
    'gamma':[0.01,0.03,0.1,0.2,0.3],
    'max_depth': [1, 5, 10, 50, 100, 500]

}
clf = RandomizedSearchCV(model, param_distributions=parameters,cv = 2,scoring = 'roc_auc',n_jobs=-1
)
print("fitting the model")
clf.fit(X_tr, y_train[:45000]["project_is_approved"])
```

```
fitting the model
```

Out[89]:

```
RandomizedSearchCV(cv=2, error_score='raise-deprecating',
          estimator=<__main__.XGBoostClassifier object at 0x0000024E4FD40940>,
          fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'eta': [0.01, 0.03, 0.05, 0.1, 0.3], 'gamma': [0.01, 0.03, 0.1, 0.2,
0.3], 'max_depth': [1, 5, 10, 50, 100, 500]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring='roc_auc', verbose=0)
```

In [91]:

```
clf.cv_results_
```

Out[91]:

```
{'mean_fit_time': array([128.47798419, 113.39395869,  34.5971818 , 126.00024843,
        126.12588859, 129.36387956, 129.06474984, 127.73326039,
        110.81338108,  47.89705241]),
 'std_fit_time': array([2.09473944, 3.18458378, 0.39197302, 2.406214  , 2.74971533,
        4.24350202, 1.75436127, 2.35676932, 0.34856737, 5.70533121]),
 'mean_score_time': array([1.90322828, 1.62870526, 1.61161053, 1.86157548, 1.58653855,
        1.82242894, 1.81296289, 1.72668576, 1.67981422, 0.92257845]),
 'std_score_time': array([0.10148573, 0.07830596, 0.07605112, 0.08926165, 0.01419997,
        0.09252763, 0.11896455, 0.11496425, 0.01720488, 0.14759028]),
 'param_max_depth': masked_array(data=[500, 50, 10, 50, 50, 100, 50, 500, 500, 500],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_gamma': masked_array(data=[0.01, 0.3, 0.01, 0.3, 0.03, 0.2, 0.1, 0.01, 0.2, 0.01],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_eta': masked_array(data=[0.1, 0.3, 0.1, 0.1, 0.01, 0.01, 0.01, 0.01, 0.03, 0.05],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 500, 'gamma': 0.01, 'eta': 0.1},
  {'max_depth': 50, 'gamma': 0.3, 'eta': 0.3},
  {'max_depth': 10, 'gamma': 0.01, 'eta': 0.1},
  {'max_depth': 50, 'gamma': 0.3, 'eta': 0.1},
  {'max_depth': 50, 'gamma': 0.03, 'eta': 0.01},
  {'max_depth': 100, 'gamma': 0.2, 'eta': 0.01},
  {'max_depth': 50, 'gamma': 0.1, 'eta': 0.01},
  {'max_depth': 500, 'gamma': 0.01, 'eta': 0.01},
  {'max_depth': 500, 'gamma': 0.2, 'eta': 0.03},
  {'max_depth': 500, 'gamma': 0.01, 'eta': 0.05}],
 'split0_test_score': array([0.61362934, 0.62243021, 0.66382804, 0.62046939, 0.59155827,
        0.59484134, 0.59239037, 0.59168987, 0.61391266, 0.61409009]),
 'split1_test_score': array([0.60989063, 0.62476105, 0.66321959, 0.61479845, 0.59803444,
        0.59844258, 0.59833504, 0.59786191, 0.60821658, 0.61002846]),
 'mean_test_score': array([0.61175999, 0.62359563, 0.66352381, 0.61763392, 0.59479635,
        0.59664196, 0.5953627 , 0.59477589, 0.61106462, 0.61205928]),
 'std_test_score': array([0.00186936, 0.00116542, 0.00030422, 0.00283547, 0.00323809,
        0.00180062, 0.00297233, 0.00308602, 0.00284804, 0.00203082]),
 'rank_test_score': array([ 5,  2,  1,  3,  9,  7,  8, 10,  6,  4]),
 'split0_train_score': array([0.9973222 , 0.99999861, 0.83000362, 0.9962772 , 0.95689321,
        0.95434474, 0.9564162 , 0.95713405, 0.97615785, 0.98669633]),
 'split1_train_score': array([0.99926679, 0.99999989, 0.81946939, 0.99881145, 0.96864913,
        0.96619715, 0.96815742, 0.96856304, 0.98105546, 0.99023448]),
 'mean_train_score': array([0.99829449, 0.99999925, 0.8247365 , 0.99754432, 0.96277117,
        0.96027094, 0.96228681, 0.96284855, 0.97860665, 0.9884654 ]),
 'std_train_score': array([9.72295112e-04, 6.39701870e-07, 5.26711249e-03, 1.26712308e-03,
        5.87796203e-03, 5.92620426e-03, 5.87061343e-03, 5.71449904e-03,
        2.44880337e-03, 1.76907179e-03])}
```

In [90]:

```
test_auc_250 = clf.cv_results_['mean_test_score']
train_auc_250 = clf.cv_results_['mean_train_score']
```

XGBOOST Model with num_boost_round = 500

In [98]:

```
model = XGBoostClassifier(eval_metric = 'auc', num_class = 2, nthread = 4,num_boost_round=500)
parameters = {
    #'num_boost_round':[5,10],
    'eta':[0.01,0.03,0.05,0.1,0.3],
    'gamma':[0.01,0.03,0.1,0.2,0.3],
    'max_depth': [3, 5, 10, 50, 100, 500]

}
clf = RandomizedSearchCV(model, param_distributions=parameters,cv = 2,scoring = 'roc_auc',n_jobs=-1
)
print("fitting the model")
clf.fit(X_tr, y_train[:45000]["project_is_approved"])
```

```
fitting the model
fitting the model
```

Out[98]:

```
RandomizedSearchCV(cv=2, error_score='raise-deprecating',
          estimator=<__main__.XGBoostClassifier object at 0x0000024E4FC61630>,
          fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'eta': [0.01, 0.03, 0.05, 0.1, 0.3], 'gamma': [0.01, 0.03, 0.1, 0.2,
0.3], 'max_depth': [3, 5, 10, 50, 100, 500]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring='roc_auc', verbose=0)
```

In [100]:

```
clf.cv_results_
```

Out[100]:

```
{'mean_fit_time': array([165.96634579, 148.85814798,  54.04647231, 167.81654632,
        146.56919944, 153.2179451 , 151.84360111, 150.5138495 ,
          9.90572119, 112.00848329]),
 'std_fit_time': array([2.8635118 , 0.68143237, 0.26828146, 3.79405129, 2.39513361,
        1.91847563, 1.28492892, 0.98502445, 0.52185583, 1.58401692]),
 'mean_score_time': array([2.4296633 , 2.00960982, 3.73862958, 1.91758299, 2.45773768,
        1.85487461, 1.95300913, 1.87243748, 1.96550679, 1.26345396]),
 'std_score_time': array([0.37725461, 0.0990926 , 0.28124762, 0.04148412, 0.26398063,
        0.12839103, 0.07184911, 0.14034629, 0.08164787, 0.03139234]),
 'param_max_depth': masked_array(data=[500, 500, 10, 100, 500, 50, 50, 500, 3, 500],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_gamma': masked_array(data=[0.01, 0.2, 0.3, 0.03, 0.1, 0.3, 0.01, 0.1, 0.1, 0.03],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_eta': masked_array(data=[0.05, 0.3, 0.05, 0.01, 0.1, 0.05, 0.05, 0.03, 0.03,
                    0.03],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 500, 'gamma': 0.01, 'eta': 0.05},
  {'max_depth': 500, 'gamma': 0.2, 'eta': 0.3},
  {'max_depth': 10, 'gamma': 0.3, 'eta': 0.05},
  {'max_depth': 100, 'gamma': 0.03, 'eta': 0.01},
  {'max_depth': 500, 'gamma': 0.1, 'eta': 0.1},
  {'max_depth': 50, 'gamma': 0.3, 'eta': 0.05},
  {'max_depth': 50, 'gamma': 0.01, 'eta': 0.05},
  {'max_depth': 500, 'gamma': 0.1, 'eta': 0.03},
  {'max_depth': 3, 'gamma': 0.1, 'eta': 0.03},
  {'max_depth': 500, 'gamma': 0.03, 'eta': 0.03}],
 'split0_test_score': array([0.61409009, 0.61362049, 0.66498766, 0.59155827, 0.61781103,
        0.61972811, 0.61409009, 0.61204145, 0.66228491, 0.60983868]),
 'split1_test_score': array([0.61002846, 0.61362172, 0.66293484, 0.59918719, 0.61247935,
        0.61335625, 0.61046137, 0.60454276, 0.65819373, 0.60760357]),
 'mean_test_score': array([0.61205928, 0.6136211 , 0.66396125, 0.59537273, 0.61514519,
        0.61654218, 0.61227573, 0.6082921 , 0.66023932, 0.60872113]),
 'std_test_score': array([2.03081633e-03, 6.13874290e-07, 1.02640978e-03, 3.81446014e-03,
        2.66584038e-03, 3.18593070e-03, 1.81436111e-03, 3.74934471e-03,
        2.04559070e-03, 1.11755667e-03]),
 'rank_test_score': array([ 7,  5,  1, 10,  4,  3,  6,  9,  2,  8]),
 'split0_train_score': array([0.98669633, 0.99999962, 0.80358184, 0.95689321, 0.9961661 ,
        0.98433589, 0.98669633, 0.97841714, 0.6695042 , 0.98031165]),
 'split1_train_score': array([0.99023448, 0.99999991, 0.79052627, 0.96824662, 0.9985647 ,
        0.9873464 , 0.99050853, 0.97872614, 0.6699868 , 0.9791261 ]),
 'mean_train_score': array([0.9884654 , 0.99999976, 0.79705406, 0.96256991, 0.9973654 ,
        0.98584115, 0.98860243, 0.97857164, 0.6697455 , 0.97971888]),
 'std_train_score': array([1.76907179e-03, 1.46943430e-07, 6.52778236e-03, 5.67670546e-03,
        1.19929648e-03, 1.50525177e-03, 1.90609641e-03, 1.54500330e-04,
        2.41296426e-04, 5.92774058e-04])}
```

In [103]:

```
test_auc_500 = clf.cv_results_['mean_test_score']
train_auc_500 = clf.cv_results_['mean_train_score']
```

In [104]:

```
test_auc = []
for i in [test_auc_5,test_auc_10,test_auc_100,test_auc_250,test_auc_500]:
    test_auc.extend(i)
train_auc = []
for i in [train_auc_5,train_auc_10,train_auc_100,train_auc_250,train_auc_500]:
    train_auc.extend(i)
```

In [113]:

```
depth= pd.Series([1,1,1,1,1,5,5,5,5,5,10,10,10,10,10,50,50,50,50,50,100,100,100,100,100,500,500,500,
500,500],index = train_auc)
splits = pd.Series([5,10,100,250,500,5,10,100,250,500,5,10,100,250,500,5,10,100,250,500,5,10,100,250
,500,5,10,100,250,500], index = train_auc)
```

3-D Scatter Plot

In [59]:

```
trace = go.Scatter3d(
    x=train_auc, y=splits, z=depth,
    mode = 'markers+text', showlegend = True,
    hovertext = ['AUC_Score','Minimum splits','Depth'],
    marker=dict(
        symbol = 'cross',
        size=8,
        color= depth,#'rgba(255,152,75,0.8)',
        colorscale='Viridis',
    ),
    line=dict(
        color='#1f77b4',
        width=1
    ),
    textfont=dict(
        family="sans serif",
        size=7,
        color="LightSeaGreen")

)
```

In [60]:

```
import plotly.graph_objects as go
fig = go.Figure(data = [trace])
fig.add_trace(go.Scatter3d(
    x=test_auc, y=splits, z=depth,
    mode = 'markers+text', showlegend = True,
    hovertext = ['AUC_Score','Minimum splits','Depth'],
    marker=dict(
        size=8,
        color= depth,#'rgba(255,152,75,0.8)',
        colorscale='Viridis',
    ),
    line=dict(
        color='#1f77b4',
        width=1
    ),
    textfont=dict(
        family="sans serif",
        size=7,
        color="LightSeaGreen")

))
fig.update_layout(title = "AUC Scores vs Depth and Splits",height = 600,showlegend = False,xaxis =
dict(title = 'AUC_SCORE'),
                  yaxis = dict(title = 'Min_Splits'))
```

In [76]:

```
model = XGBoostClassifier(eval_metric = 'auc', num_class = 2, nthread = 4,num_boost_round=5,max_dep
th = 5,eta =  0.1,gamma = 0.03)
model.fit(X_tr,y_train[:45000]["project_is_approved"])
```

In [63]:

```
# batch wise prediction
def proba_predict(model , data):
    y_pred_data = []
    n_loop = data.shape[0] - data.shape[0]%1000
    # here 1000 represents batch_size
    for i in range(0,n_loop,1000):
        y_pred_data.extend(model.predict_proba(data[i:i+1000])[:,1])
    if data.shape[0]%1000!=0:
        y_pred_data.extend(model.predict_proba(data[n_loop:])[:,1])
    return(y_pred_data)
```

In [77]:

```
y_train_pred = proba_predict(model,X_tr)
y_test_pred = proba_predict(model,X_ts)

fpr_train,tpr_train,thres_train = roc_curve(y_train[:45000]["project_is_approved"], y_train_pred)
fpr_test,tpr_test,thres_test = roc_curve(y_test[:15000]["project_is_approved"], y_test_pred)

fig = go.Figure()
fig.add_trace(go.Scatter(x = fpr_train,y = tpr_train,name='Train_AUC',text = "Train AUC Score ="+st
r(auc(fpr_train, tpr_train))))
fig.add_trace(go.Scatter(x = fpr_test,y = tpr_test,name = "Test_AUC",text = "Test AUC Score ="+str(
auc(fpr_test, tpr_test))))
fig.add_trace(go.Scatter(x = np.linspace(0,1,600),y = np.linspace(0,1,600),name = '0.5 AUC Score'))

fig.update_layout(title = 'ROC_AUC SCORE',
                  xaxis = go.layout.XAxis(title = go.layout.xaxis.Title(text = 'True Positive Rate
(TPR)')),
                 yaxis = go.layout.YAxis(title = go.layout.yaxis.Title(text = "False Positive Rate
(FPR)")))
fig.show()
```

In [81]:

```
model = XGBoostClassifier(eval_metric = 'auc', num_class = 2, nthread = 4,num_boost_round=10,max_de
pth = 5,gamma = 0.2,eta = 0.05)
print("Fitting the model")
model.fit(X_tr,y_train[:45000]["project_is_approved"])
```

```
Fitting the model
```

In [82]:

```
y_train_pred = proba_predict(model,X_tr)
y_test_pred = proba_predict(model,X_ts)

fpr_train,tpr_train,thres_train = roc_curve(y_train[:45000]["project_is_approved"], y_train_pred)
fpr_test,tpr_test,thres_test = roc_curve(y_test[:15000]["project_is_approved"], y_test_pred)

fig = go.Figure()
fig.add_trace(go.Scatter(x = fpr_train,y = tpr_train,name='Train_AUC',text = "Train AUC Score ="+st
r(auc(fpr_train, tpr_train))))
fig.add_trace(go.Scatter(x = fpr_test,y = tpr_test,name = "Test_AUC",text = "Test AUC Score ="+str(
auc(fpr_test, tpr_test))))
fig.add_trace(go.Scatter(x = np.linspace(0,1,600),y = np.linspace(0,1,600),name = '0.5 AUC Score'))

fig.update_layout(title = 'ROC_AUC SCORE',
                  xaxis = go.layout.XAxis(title = go.layout.xaxis.Title(text = 'True Positive Rate
(TPR)')),
                 yaxis = go.layout.YAxis(title = go.layout.yaxis.Title(text = "False Positive Rate
(FPR)")))
fig.show()
```

In [108]:

```
model = XGBoostClassifier(eval_metric = 'auc', num_class = 2, nthread = 4,num_boost_round=10,max_depth = 3,gamma = 0.2,eta = 0.05)
print("Fitting the model")
model.fit(X_tr,y_train[:45000]["project_is_approved"])
```

```
Fitting the model
```

In [109]:

```
y_train_pred = proba_predict(model,X_tr)
y_test_pred = proba_predict(model,X_ts)

fpr_train,tpr_train,thres_train = roc_curve(y_train[:45000]["project_is_approved"], y_train_pred)
fpr_test,tpr_test,thres_test = roc_curve(y_test[:15000]["project_is_approved"], y_test_pred)

fig = go.Figure()
fig.add_trace(go.Scatter(x = fpr_train,y = tpr_train,name='Train_AUC',text = "Train AUC Score ="+str(auc(fpr_train, tpr_train))))
fig.add_trace(go.Scatter(x = fpr_test,y = tpr_test,name = "Test_AUC",text = "Test AUC Score ="+str(auc(fpr_test, tpr_test))))
fig.add_trace(go.Scatter(x = np.linspace(0,1,600),y = np.linspace(0,1,600),name = '0.5 AUC Score'))

fig.update_layout(title = 'ROC_AUC SCORE',
                  xaxis = go.layout.XAxis(title = go.layout.xaxis.Title(text = 'True Positive Rate (TPR)')),
                 yaxis = go.layout.YAxis(title = go.layout.yaxis.Title(text = "False Positive Rate (FPR)")))
fig.show()
```

In [87]:

```
model = XGBoostClassifier(eval_metric = 'auc', num_class = 2, nthread = 4,num_boost_round=100,max_d
epth = 5,gamma = 0.03,eta=0.1)
print("Fitting the model")
model.fit(X_tr,y_train[:45000]["project_is_approved"])
```

```
Fitting the model
```

In [88]:

```
y_train_pred = proba_predict(model,X_tr)
y_test_pred = proba_predict(model,X_ts)

fpr_train,tpr_train,thres_train = roc_curve(y_train[:45000]["project_is_approved"], y_train_pred)
fpr_test,tpr_test,thres_test = roc_curve(y_test[:15000]["project_is_approved"], y_test_pred)

fig = go.Figure()
fig.add_trace(go.Scatter(x = fpr_train,y = tpr_train,name='Train_AUC',text = "Train AUC Score ="+st
r(auc(fpr_train, tpr_train))))
fig.add_trace(go.Scatter(x = fpr_test,y = tpr_test,name = "Test_AUC",text = "Test AUC Score ="+str(
auc(fpr_test, tpr_test))))
fig.add_trace(go.Scatter(x = np.linspace(0,1,600),y = np.linspace(0,1,600),name = '0.5 AUC Score'))

fig.update_layout(title = 'ROC_AUC SCORE',
                  xaxis = go.layout.XAxis(title = go.layout.xaxis.Title(text = 'True Positive Rate
(TPR)')),
                 yaxis = go.layout.YAxis(title = go.layout.yaxis.Title(text = "False Positive Rate
(FPR)")))
fig.show()
```

In [110]:

```
model = XGBoostClassifier(eval_metric = 'auc', num_class = 2, nthread = 4,num_boost_round=100,max_depth = 3,gamma = 0.03,eta=0.1)
print("Fitting the model")
model.fit(X_tr,y_train[:45000]["project_is_approved"])
```

```
Fitting the model
```

In [111]:

```
y_train_pred = proba_predict(model,X_tr)
y_test_pred = proba_predict(model,X_ts)

fpr_train,tpr_train,thres_train = roc_curve(y_train[:45000]["project_is_approved"], y_train_pred)
fpr_test,tpr_test,thres_test = roc_curve(y_test[:15000]["project_is_approved"], y_test_pred)

fig = go.Figure()
fig.add_trace(go.Scatter(x = fpr_train,y = tpr_train,name='Train_AUC',text = "Train AUC Score ="+str(auc(fpr_train, tpr_train))))
fig.add_trace(go.Scatter(x = fpr_test,y = tpr_test,name = "Test_AUC",text = "Test AUC Score ="+str(auc(fpr_test, tpr_test))))
fig.add_trace(go.Scatter(x = np.linspace(0,1,600),y = np.linspace(0,1,600),name = '0.5 AUC Score'))

fig.update_layout(title = 'ROC_AUC SCORE',
                  xaxis = go.layout.XAxis(title = go.layout.xaxis.Title(text = 'True Positive Rate (TPR)')),
                 yaxis = go.layout.YAxis(title = go.layout.yaxis.Title(text = "False Positive Rate (FPR)")))
fig.show()
```

In [96]:

```
model = XGBoostClassifier(eval_metric = 'auc', num_class = 2, nthread = 4,num_boost_round=250,max_depth = 3,gamma = 0.01,eta=0.1)
print("Fitting the model")
model.fit(X_tr,y_train[:45000]["project_is_approved"])
```

```
Fitting the model
```

In [97]:

```
y_train_pred = proba_predict(model,X_tr)
y_test_pred = proba_predict(model,X_ts)

fpr_train,tpr_train,thres_train = roc_curve(y_train[:45000]["project_is_approved"], y_train_pred)
fpr_test,tpr_test,thres_test = roc_curve(y_test[:15000]["project_is_approved"], y_test_pred)

fig = go.Figure()
fig.add_trace(go.Scatter(x = fpr_train,y = tpr_train,name='Train_AUC',text = "Train AUC Score ="+st
r(auc(fpr_train, tpr_train))))
fig.add_trace(go.Scatter(x = fpr_test,y = tpr_test,name = "Test_AUC",text = "Test AUC Score ="+str(
auc(fpr_test, tpr_test))))
fig.add_trace(go.Scatter(x = np.linspace(0,1,600),y = np.linspace(0,1,600),name = '0.5 AUC Score'))

fig.update_layout(title = 'ROC_AUC SCORE',
                  xaxis = go.layout.XAxis(title = go.layout.xaxis.Title(text = 'True Positive Rate
(TPR)')),
                 yaxis = go.layout.YAxis(title = go.layout.yaxis.Title(text = "False Positive Rate
(FPR)")))
fig.show()
```

In [106]:

```
model = XGBoostClassifier(eval_metric = 'auc', num_class = 2, nthread = 4,num_boost_round=500,max_d
epth = 3,gamma = 0.1,eta = 0.03)
print("Fitting the model")
model.fit(X_tr,y_train[:45000]["project_is_approved"])
```

```
Fitting the model
```

In [107]:

```
y_train_pred = proba_predict(model,X_tr)
y_test_pred = proba_predict(model,X_ts)

fpr_train,tpr_train,thres_train = roc_curve(y_train[:45000]["project_is_approved"], y_train_pred)
fpr_test,tpr_test,thres_test = roc_curve(y_test[:15000]["project_is_approved"], y_test_pred)

fig = go.Figure()
fig.add_trace(go.Scatter(x = fpr_train,y = tpr_train,name='Train_AUC',text = "Train AUC Score ="+st
r(auc(fpr_train, tpr_train))))
fig.add_trace(go.Scatter(x = fpr_test,y = tpr_test,name = "Test_AUC",text = "Test AUC Score ="+str(
auc(fpr_test, tpr_test))))
fig.add_trace(go.Scatter(x = np.linspace(0,1,600),y = np.linspace(0,1,600),name = '0.5 AUC Score'))
fig.add_trace(go.Scatter(x = np.linspace(0,1,600),y = np.linspace(0,1,600),name = '0.5 AUC Score'))

fig.update_layout(title = 'ROC_AUC SCORE',
                  xaxis = go.layout.XAxis(title = go.layout.xaxis.Title(text = 'True Positive Rate
(TPR)')),
                 yaxis = go.layout.YAxis(title = go.layout.yaxis.Title(text = "False Positive Rate
(FPR)")))
fig.show()
```

3. Conclusion

In [112]:

```
from prettytable import PrettyTable

#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable

x = PrettyTable()
x.field_names = ["Model", "Max_Depth","Min_Number_split","Eta","lambda","BEST_AUC_SCORE"]
x.add_row(["TruncatedSVD",5 ,5,0.03,0.1,0.6240])
x.add_row(["TruncatedSVD",5 ,10,0.05,0.2,0.6290])
x.add_row(["TruncatedSVD",3 ,10,0.05,0.2,0.6323])
x.add_row(["TruncatedSVD",5 ,100,0.03,0.1,0.6182])
x.add_row(["TruncatedSVD",3 ,100,0.03,0.1,0.6391])
x.add_row(["TruncatedSVD",3 ,250,0.01,0.1,0.6244])
x.add_row(["TruncatedSVD",3 ,500,0.03,0.1,0.6398])



print(x)
```

```
+--------------+-----------+------------------+------+--------+----------------+
|    Model     | Max_Depth | Min_Number_split | Eta  | lambda | BEST_AUC_SCORE |
+--------------+-----------+------------------+------+--------+----------------+
| TruncatedSVD |     5     |        5         | 0.03 |  0.1   |     0.624      |
| TruncatedSVD |     5     |        10        | 0.05 |  0.2   |     0.629      |
| TruncatedSVD |     3     |        10        | 0.05 |  0.2   |     0.6323     |
| TruncatedSVD |     5     |       100        | 0.03 |  0.1   |     0.6182     |
| TruncatedSVD |     3     |       100        | 0.03 |  0.1   |     0.6391     |
| TruncatedSVD |     3     |       250        | 0.01 |  0.1   |     0.6244     |
| TruncatedSVD |     3     |       500        | 0.03 |  0.1   |     0.6398     |
+--------------+-----------+------------------+------+--------+----------------+
```